The Multidimensional Knapsacks problem with Family-Split Penalties (MKFSP)

R.Mansini, L. Moreschini

29 September 2022

1 Problem statement

1.1 Definition

Given $n, m \in \mathbb{N}^+$, let $I = \{1, \dots, n\}$ be a set of items, let $F = \{1, \dots m\}$ and let $\{F_j\}_{j \in F}$ be a partition of I, i.e.

$$\begin{aligned} F_j &\neq \emptyset, && \forall j \in F, \\ F_j \cap F_l &= \emptyset, && \forall j, l \in F : j \neq l, \\ \bigcup_{j \in F} F_j &= I. \end{aligned}$$

Given $k_{\max}, r_{\max} \in \mathbb{N}^+$, let $K = \{1, \dots, k_{\max}\}$ be the set of available knapsacks, each having a maximum capacity $C_{kr} \in \mathbb{N}$ for each resource type $r \in R = \{1, \dots, r_{\max}\}$. Each item $i \in I$ requires an amount $w_{ir} \in \mathbb{N}$ of resource $r \in R$ to be allocated into knapsack $k \in K$. The Multidimensional Knapsacks problem with Family-Split Penalties (MKFSP) requires to find the best assignment(s) of the items among the available knapsacks in order to maximize the total value under the following conditions.

1. If an item $i \in F_j$ $(j \in F)$ is selected, then all the items of the family F_j must be selected,
2. the items of any family may be assigned to different knapsacks,
3. each family of items may or may not be selected,
4. knapsacks capacities for each resource $r \in R$ can not be exceeded,
5. selecting family $j \in F$ provides a given profit $p_j \in \mathbb{N}$,
6. each time a family $j \in F$ is split among more than one knapsack, a penalty δ_j has to be paid.

1.2 ILP model

The proposed ILP model makes use of the following variables.

- $x_j = \begin{cases} 1 & \text{if family } j \in F \text{ is selected,} \\ 0 & \text{otherwise.} \end{cases}$
- $y_{ik} = \begin{cases} 1 & \text{if item } i \in I \text{ is allocated to knapsack } k \in K, \\ 0 & \text{otherwise.} \end{cases}$
- $z_{jk} = \begin{cases} 1 & \text{if at least one item of family } j \in F \\ & \text{is allocated to knapsack } k \in K, \\ 0 & \text{otherwise.} \end{cases}$
- $s_j \in \mathbb{N}$: how many times the family $j \in F$ has been split among more than one knapsack.

The proposed ILP formulation is as follows.

$$\max \quad \sum_{j \in F} (p_j x_j - \delta_j s_j), \tag{1}$$

$$\sum_{k \in K} y_{ik} = x_j, \qquad \forall j \in F,\ \forall i \in F_j, \tag{2}$$

$$\sum_{i \in I} w_{ir} y_{ik} \le C_{kr}, \qquad \forall k \in K,\ \forall r \in R, \tag{3}$$

$$\sum_{i \in F_j} y_{ik} \le |F_j| z_{jk}, \qquad \forall j \in F,\ \forall k \in K, \tag{4}$$

$$\sum_{k \in K} z_{jk} - 1 \le s_j, \qquad \forall j \in F, \tag{5}$$

$$x_j,\ y_{ik},\ z_{jk} \in \{0, 1\}, \qquad \forall i \in I,\ \forall j \in F,\ \forall k \in K, \tag{6}$$

$$s_j \in \mathbb{N}, \qquad \forall j \in F. \tag{7}$$

The model maximizes the sum of the profits of the selected families minus the sum of the penalties due to the splitting of a family among more than one knapsack. Constraints (2) ensure that each item is loaded in exactly one knapsack if and only if the family it belongs to is selected (loaded among one or more knapsacks). Constraints (3) ensure that for each knapsack its maximum capacity for each resource type is not exceeded. Constraints (4) of type "BigM" set the value of variables z_{jk} ($j \in F$, $k \in K$) to 1 if at least one item $i \in F_j$ is assigned to knapsack k. Constraints (5) set the value of variables s_j ($j \in F$) equal to the numer of times each family is split among more than one knapsack Both constraints (4) and (5) relay on the objective function maximization sense together with δ_j being positive values. Finally, (6) and (7) define the domain of each variable.

2 Assignment

2.1 Tasks

Your primary task is to *develop* and *implement* an heuristic algorithm to solve the MKFSP problem on the provided benchmark instances. ‘Develop’ means that your algorithm must be thoroughly described and documented with pseudocode in a section of your final report. ‘Implement’ means that you also have to write a program that follows the abstract description of your algorithm to solve the provided benchmark instances. Your algorithm can be implemented either in Java or in Python.

In addition you will also have to *gather* and *compare* information about the performance of the proposed ILP model solved with Gurobi and your algorithm. ‘Gather’ means that you will have to collect some information about the execution and the runtime behaviour of both strategies. You will have to keep track (at least) of the following ones (feel free to add more as you see fit).

- status
- time limit (if any)
- start at
- end at
- runtime
- time to best
- model building time (Gurobi)
- solution
- solution objective value
- ratio of loaded families
- ratio of loaded items
- total penalties paid
- free space left in each knapsack

Such information must be provided in a separate file for each benchmark instance, either in `CSV` or `JSON` format. ‘Compare’ means that inside your final report you must discuss and constrast the gathered information, possibly with reference to the different carachteristics of each benchmark instance (e.g., number of items, number of knapsacks, number of families, . . .).

Finally, as an optional task, you are also welcome to analyse the formulation of the proposed ILP model, suggest improvements or even propose a different (possibly stronger) ILP formulation.

2.2 Deliverables

Your final submission must include the source code of your algorithm, one data file for each benchmark instance containg information about the behaviour of Gurobi and your algorithm (in `CSV` or `JSON` format) and a written report. The source code must be made available in a `git` repository that the examiners must be allowed to monitor throughout the development of the project. The delivered source code must be executable by the examiners and its results must be similar to those contained in the delivered data files. The report must be a pdf document (preferably made with LaTeX) with the following structure.

- 'Introduction' section: TODO: breve descrizione di cosa scrivere, at most 2 pages,
- 'Heuristic algorithm' section: TODO: breve descrizione di cosa scrivere, at most 10 pages,
- 'Benchmark instances evaluation' section: TODO: breve descrizione di cosa scrivere, at most 5 pages,
- 'Conclusion' section: TODO: breve descrizione di cosa scrivere, at most 2 pages.

3 Benchmark instances

Important: although in the formal statement of the problem indexes start from 1, all benchmark instances use 0-based indexes (as all array-like data structures in Java and Python use 0-based indexes). For example, given a list of n items, the first item has index 0, the second item has index 1 and so on... the last item has index $n-1$.

3.1 Format description

In each instance file items are sorted by family: if the first family has size f_1, its items are indexed from 0 to $f_1 - 1$, f_1 is the index of the first item of the second family and so on. Benchmark instances are stored in text files with `JSON` format. Each file describes an instance and contains a JSON object with the following properties.

- `id`: a string, a name that uniquely identifies the instance,
- `n_items`: an integer, the number of available items, i.e. n,
- `n_families`: an integer, the number of available families, i.e. m.
- `n_knapsacks`: an integer, the number of available items, i.e. $k_{\max}$,
- `n_resources`: an integer, the number of available resources, i.e. $r_{\max}$,
- `profits`: an array of integers of length m, the value in position $0 \leq j \leq m-1$ is the profit obtained when family j is selected,
- `penalties`: an array of integers of length m, the value in position $0 \leq j \leq m-1$ is the penalty paid each time family j is split among more than on knapsack,

- `first_items`: an array of integers of length m, the value in position $0 \leq j \leq m-1$ is the index of the first item that belongs to family j,
- `items`: an array of length n of arrays of length $r_{\max}$ of integers, for $0 \leq i \leq n-1$ and $0 \leq r \leq r_{\max}-1$ the value of `items[i][r]` equals w_{ir}, i.e. the amount of resource of type r required to assign item i in a knapsack,
- `kanpsacks`: an array of length $k_{\max}$ of arrays of length $r_{\max}$ of integers, for $0 \leq k \leq k_{\max}-1$ and $0 \leq r \leq r_{\max}-1$ the value of `knapsacks[i][r]` equals C_{kr}, i.e. the maximum capacity of knapsack k with respect to resource type r.

3.2 Example

Listing 1 shows the data format of a simple instance with 12 items, 4 families, 3 knapsacks and 2 types of resources.
The first family has a profit of 10 and a penalty of 4 for each split, the second family has a profit of 20 and a penalty of 3 for each split, the third family has a profit of 30 and a penalty of 2 for each split and the fourth family has a profit of 40 and a penalty of 1 for each split.
Items 0, 1, 2 and 3 belong to the first family, items 4 and 5 belong to the second family, items 6, 7, 8, 9 and 10 belong to the third family and the fourth family contains only item number 11.
Item number 0 requires 1 unit of the first resource type and 2 units of the second resource type, item number 1 requires 3 units of the first resource type and 4 units of the second resource type, item number 2 requires 8 units of the first resource type and 6 units of the second resource type, and so on.
Finally, knapsack number 0 has a maximum capacity of 10 units for the first resource type and 10 units for the second resource type, knapsack number 1 has a maximum capacity of 10 units for the first resource type and 5 units for the second resource type, knapsack number 2 has a maximum capacity of 15 units for the first resource type and 15 units for the second resource type.

```
{
    "id": "example",
    "n_items": 12,
    "n_families": 4,
    "n_knapsacks": 3,
    "n_resources": 2,
    "profits": [10, 20, 30, 40],
    "penalties": [4, 3, 2, 1],
    "first_items": [0, 4, 6, 11],
    "items": [
        [1, 2], [3, 4], [8, 6], [5, 5],
        [3, 3], [7, 1],
        [2, 0], [0, 7], [3, 2], [3, 1], [1, 8],
        [4, 5]
    ],
    "knapsacks": [
        [10, 10],
        [10,  5],
        [15, 15]
    ]
}
```

Listing 1: Content of instance file `example.json`.